HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD

www.heig-vd.ch

05 - Persistence tier (2)

Transactions with EJBs and JDBC

AMT 2018

Olivier Liechti

Two topics for this week

Transactions with EJBs and JDBC
Automated User Acceptance Tests (UAT)

Transactions

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Imagine that you have the following code in a business service:

```
accountA.debit(100);
accountB.credit(100);
```

What happens is the application crashes here?
Is my data corrupted?
Has money vanished in cyberspace?

Transactions

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

```
transaction.start();

accountA.debit(100);
accountB.credit(100);

transaction.commit();
```

Transactions give us an “whole or nothing” semantic
(we often speak about a unit of work)

Transactions

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

```
transaction.start();
accountA.debit(100);
try {
  accountB.credit(100);
} catch (AccountFullException e) {
  transaction.rollback();
}
transaction.commit();
```

We can also deal with application-level errors and leave the data in a consistent state.

Transactions

ACID

Transactions

ACID

Atomicity: “all or noting”

Transactions

ACID

Consistency: “business data integrity”

Transactions

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

ACID

Isolation: “deal with concurrent transactions”

There are different isolation levels!

Isolation levels

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Increasing isolation between transactions

Isolation level	Potential issues
Read Uncommitted (no locks)	**Dirty Reads** (no isolation)
Read Committed (write locks)	**Non-repeatable Reads**
Repeatable Reads (read & write locks)	**Phantom reads**
Serializable (range locks)	

Increasing performance in the cas of concurrent access

Isolation levels

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

- "A **dirty read** occurs when a transaction is allowed to read data from a row that has been modified by another running transaction and not yet committed."
- "A **non-repeatable read** occurs, when during the course of a transaction, a row is retrieved twice and the values within the row differ between reads."
- "A **phantom read** occurs when, in the course of a transaction, two identical (SELECT) queries are executed, and the **collection** of rows returned by the second query is different from the first."

See for **example scenarios**, see:

https://docs.oracle.com/javase/tutorial/jdbc/basics/transactions.html#transactions_data_integrity

https://en.wikipedia.org/wiki/Isolation_(database_systems)#Read_phenomena

Transactions

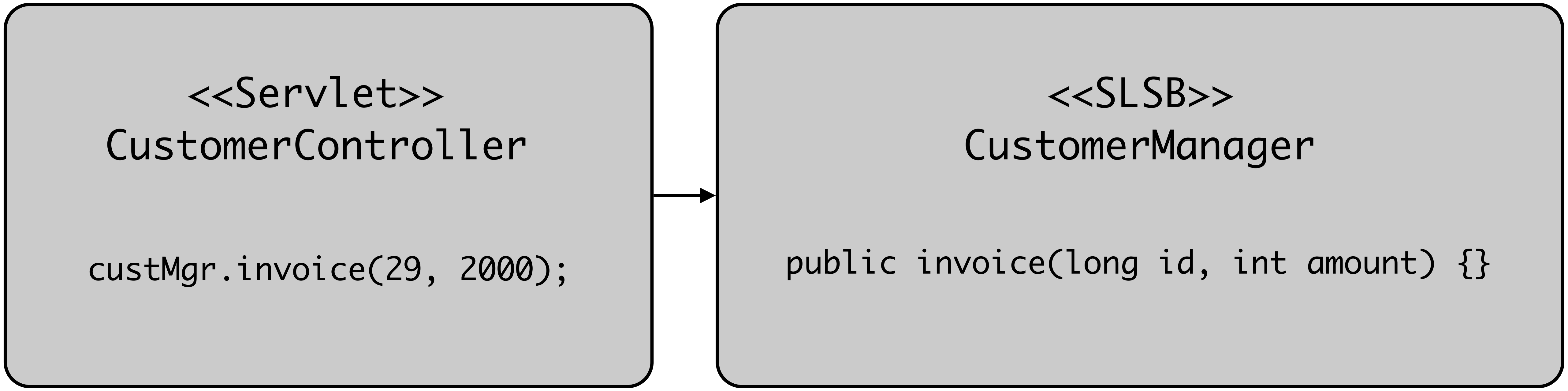
heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

ACID

Durability: “once it’s done, it’s done”

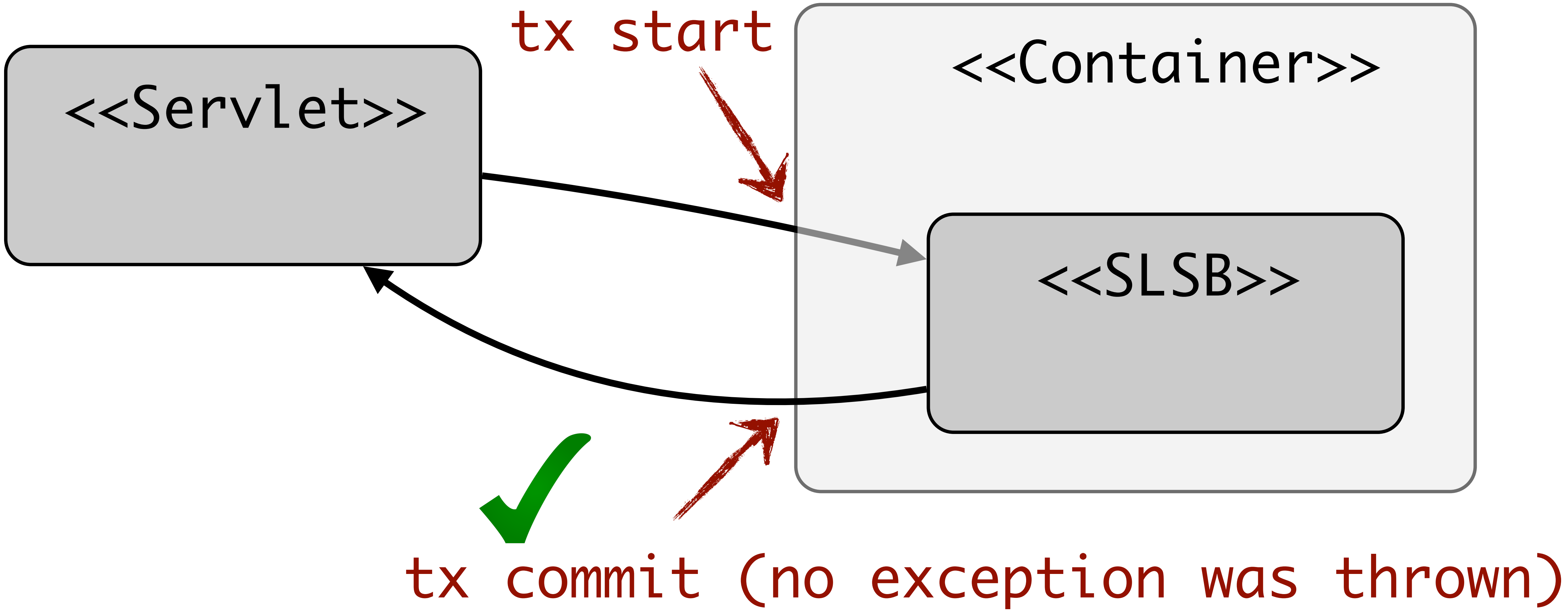
Transactions & EJBs

- By default, the EJB container handles calls to commit and rollback.
- Methods defined on EJBs provide demarcation points.
- This is the **default behavior**.


<<Servlet>>
CustomerController
custMgr.invoice(29, 2000);
<<SLSB>>
CustomerManager
public invoice(long id, int amount) {}

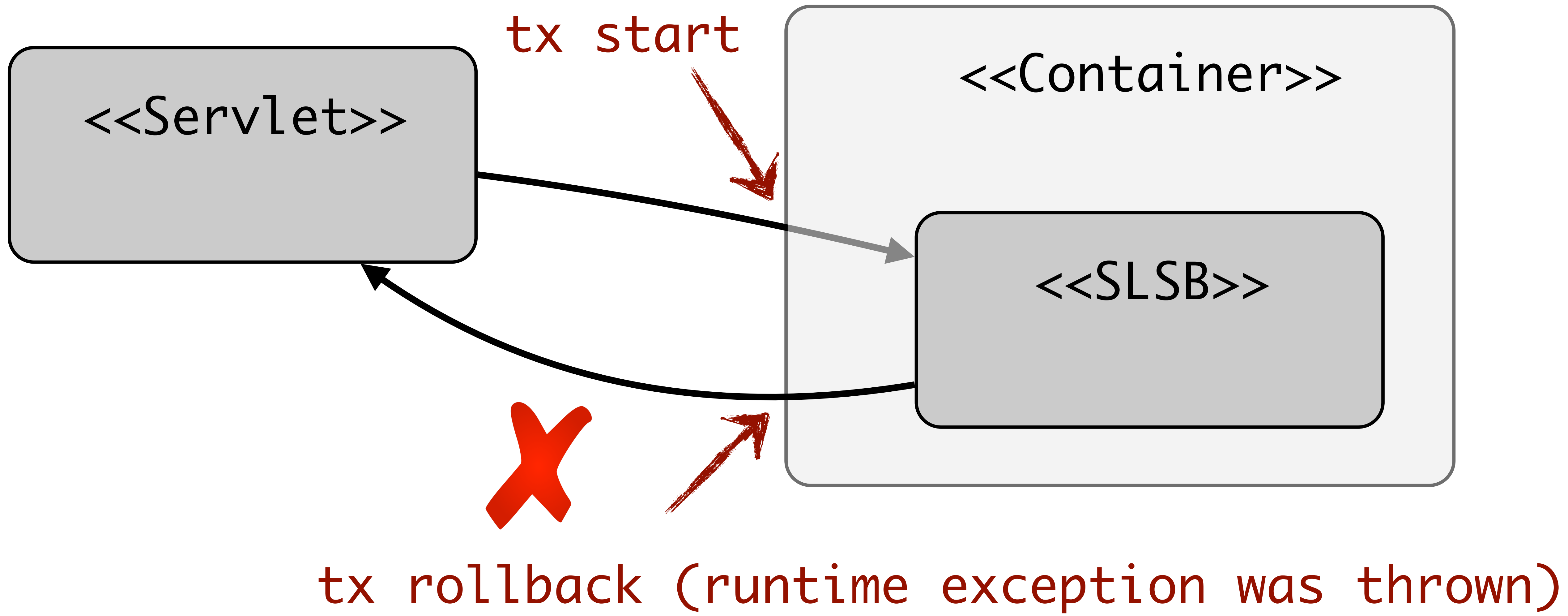
Transactions & EJBs

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud



Transactions & EJBs

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud



Transaction Scope

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

What happens when a **client** calls a method on a session bean,

which calls a method on a session bean,
which calls a method on a session bean,
which calls a method on a session bean,
which calls a method on a session bean,
which calls a method on a session bean,
which calls a method on a session bean,

which **throws an exception**?

Transaction Scope

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

What happens when a **client** calls a method on a sess

session bean,
whi on a session bean,
which calls a method on a session bean,
which calls a method on a
which calls a method o
which calls a method o

which **throws an exception**?

Opinion1
Everything should be rolled back!

Opinion2
No! Only changes incurred by the last method should be rolled back!

Transaction Scope

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

What happens when a **client** calls a method on a sess

whic

which calls a method

which calls a method on a

which calls a method o

which calls a method o

which **throws an exception**?

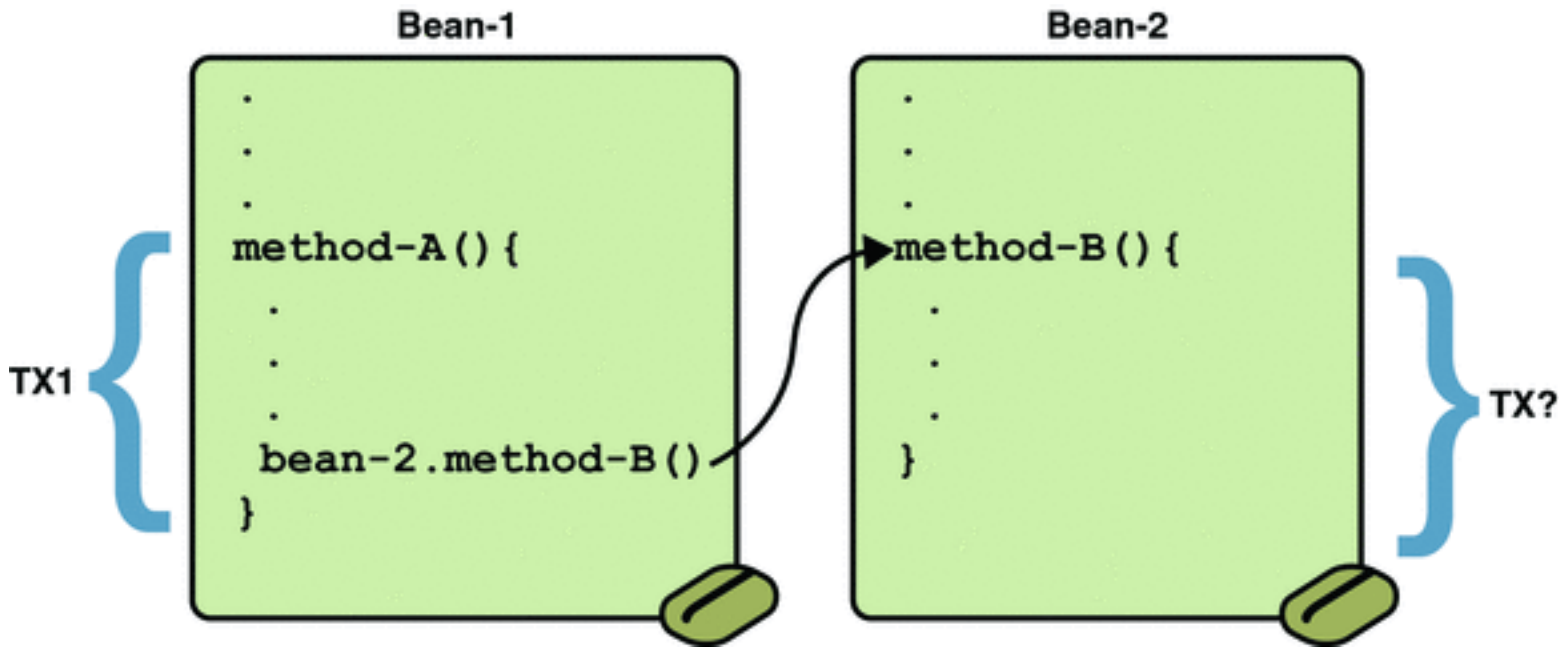
Opinion1
Everything should rolled back!

It is **up to the application** to specify intended behavior. The developer must specify transaction scope, typically with **annotations**.

Opinion2
No! Only changes incurred by the last method should be rolled back!

Transaction Scope

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud



http://java.sun.com/javaee/5/docs/tutorial/doc/bncij.html

Transaction Scope

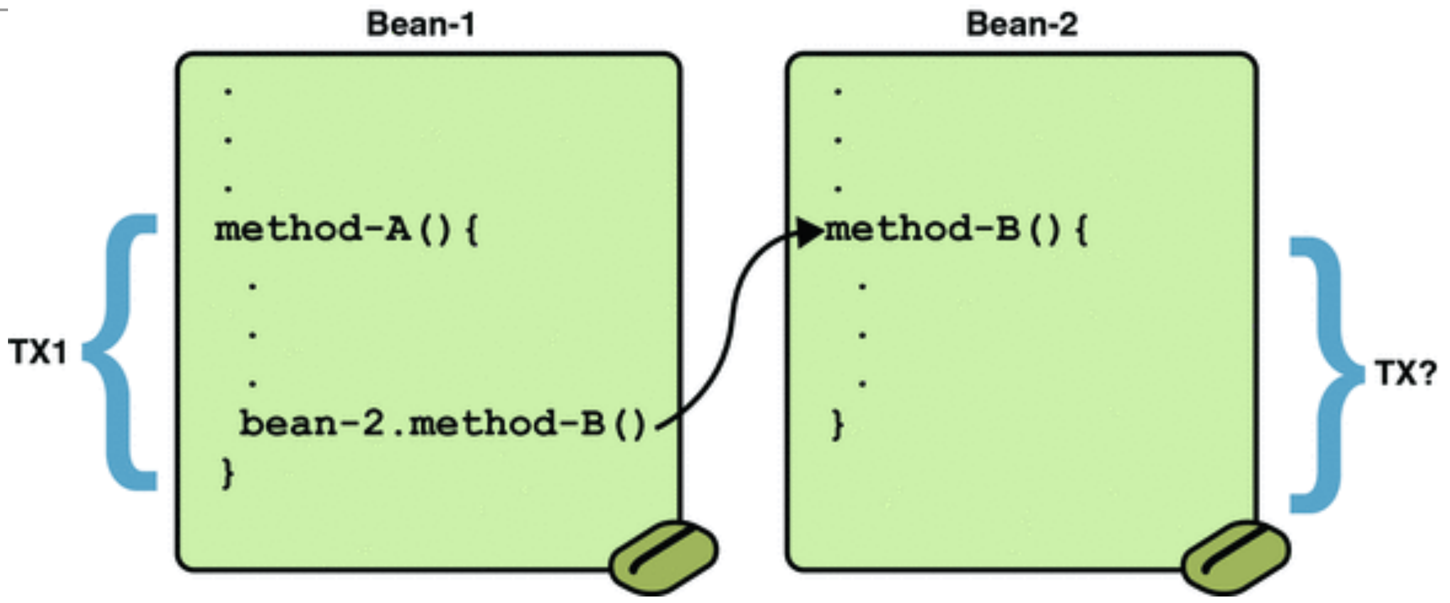
heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

```
@TransactionAttribute(NOT_SUPPORTED)
@Stateless
public class TransactionBean implements
Transaction {
...
    @TransactionAttribute(REQUIRES_NEW)
    public void firstMethod() {...}

    @TransactionAttribute(REQUIRED)
    public void secondMethod() {...}

    public void thirdMethod() {...}

    public void fourthMethod() {...}
}
```



Transaction Attribute	Client's Transaction	Business Method's Transaction
Required	None	T2
	T1	T1
RequiresNew	None	T2
	T1	T2
Mandatory	None	error
	T1	T1
NotSupported	None	None
	T1	None
Supports	None	None
	T1	T1
Never	None	None
	T1	Error

Transactions & Exceptions

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

- There are **two ways to roll back a container-managed transaction**
- Firstly, if a **system exception** is thrown, the container will automatically roll back the transaction.
- Secondly, by invoking the **setRollbackOnly** method of the EJBContext interface, the bean method instructs the container to roll back the transaction.
- If the bean throws an **application exception**, the rollback is not automatic but can be initiated by a call to **setRollbackOnly**.
- Note: you can also annotate your Exception class with **@ApplicationException(rollback=true)**